APPLICATION AND COMPARISON OF

Machine Learning Algorithms for Steel Quality Prediction

Diego Esteban Zapata Sanchez
Matrikel nummer m12317103

CONTENTS

1. OBJECTIVES
2. ACCESS AND PREPROCESS THE DATA
   - ✓ Import data
   - ✓ Transform data
   - ✓ Drop duplicated rows
   - ✓ Handle null values
   - ✓ Remove outliers
3. EXPLORATORY DATA ANALYSYS THROUGH VISUALIZATION
   - Box plots
   - Scatter plots
   - Heat maps
   - Histograms
4. CROSS VALIDATION AND HYPERPARAMETERS
5. **MODEL ARCHITECTURE - MACHINE LEARNING ALGORITHMS.**
   - Multiple Regression
   - Random Forest
   - Support Vector Machine
   - Neural Network

OBJECTIVES

General objective:

Predict a quality relevant metric (yield strength) through the use of different machine learning algorithms, training the models with the given dataset of sensor and process data.

Specific objectives:

- Implement effective data set cleaning techniques and save the processed data.
- Make and sabe graphs to visualize the distribution and trends of the data.
- Perform analysis of the data set based on its statistics and graphs.
- Develop machine learning models to predict steel quality from the continuous casting plants dataset provided.
- Train the model to learn and map the input process data to the target steel quality variables.
- Compare various ML architectures to identify the most effective model for accurate steel quality prediction.
- Assess the performance of implemented models using appropriate evaluation metrics.

ACCESS AND PREPROCESS THE DATA

```python
def shuffle_data(df1, df2):
    """...
    df_combined = pd.concat([df1, df2], ignore_index = True)

    # Frac = 1 corresponds to 100% of the data. drop = True corresponds to not preserving old indexes.
    df_combined_shuffled = df_combined.sample(frac = 1, random_state = 42).reset_index(drop = True)

    return df_combined_shuffled


def separate_data_frame(df, percentage_training = 75, percentage_testing = 15):
    """...
    total_samples = len(df)
    train_size = int((percentage_training/100)*total_samples)
    test_size = int((percentage_testing/100)*total_samples)

    df_train = df[:train_size]
    df_test = df[train_size:(train_size + test_size)]
    df_validation = df[(train_size + test_size):]

    return df_train, df_test, df_validation
```

Import data

```python
general_path = 'C:/Users/di_estebannn/Desktop/universidad/austria/applied_machine_and_deep_learning/project'
processed_data_path = os.path.join(general_path, 'data', 'processed')
file_path_train = general_path + '/data/raw/normalized_train_data.csv'
file_path_test = general_path + '/data/raw/normalized_test_data.csv'

df_train_data = pd.read_csv(file_path_train)
df_test_data = pd.read_csv(file_path_test)

df_total_data = shuffle_data(df_train_data, df_test_data)
X, y = separate_and_clean_data('Total data set', df_total_data, already_cleaned = True)
save_data((general_path + '/data/raw/'), 'normalized_total_data.csv', X, y)

df_train, df_test, df_validation = separate_data_frame(df_total_data)
```

✓ 0.8s

```
Sizes of the Total data set: X = (10979, 21), y = (10979,).

The normalized_total_data.csv has been saved in C:/Users/di_estebannn/Desktop/universidad/austria/applied_machin
```

ACCESS AND PREPROCESS THE DATA

Transform data and Drop duplicated rows

```python
X_train, y_train = separate_and_clean_data('Training set', df_train)
X_test, y_test = separate_and_clean_data('Testing set', df_test)
X_validation, y_validation = separate_and_clean_data('Validation set', df_validation)
X, y = separate_and_clean_data('Total data set', df_total_data)
```

✓ 0.0s

```
Sizes of the Training set: X = (8234, 21), y = (8234,).

Sizes of the Testing set: X = (1646, 21), y = (1646,).

izes of the Validation set: X = (1099, 21), y = (1099,).

Sizes of the Total data set: X = (10979, 21), y = (10979,).
```

```python
def separate_and_clean_data(name, data_frame, already_cleaned = False):
    """...
    if already_cleaned == False:
        # Convert all data to numeric type (or null if not possible) and
        data_frame = data_frame.apply(pd.to_numeric, errors = 'coerce')
        data_frame = data_frame.dropna(subset = ['output'])
        data_frame.drop_duplicates(inplace = True)

    # Separate the data frame into X (inputs) and y (outputs) and see th
    X = data_frame.iloc[:, 1:]
    y = data_frame.iloc[:, 0]
    print(f'Sizes of the {name}: X = {X.shape}, y = {y.shape}.\n')

    return X, y
```

ACCESS AND PREPROCESS THE DATA

Handle null values

```python
X_train, means = fill_null_values(X_train, y_train, 'Training set')
X_test, _ = fill_null_values(X_test, y_test, 'Testing set', means)
X_validation, _ = fill_null_values(X_validation, y_validation, 'Validation set')
X, _ = fill_null_values(X, y, 'Total data set')
```

✓ 0.4s

```
The total amount of data for the Training set is equal to 8234.0 in all columns.
It is False that there are null values, and the means of the inputs in the Training set is
[[0.59527569 0.80519772 0.73970471]
 [0.43339199 0.58410562 0.39352117]
 [0.41149113 0.67946258 0.51885498]
 [0.11483995 0.06679987 0.53295611]
 [0.2470088  0.26888976 0.19573641]
 [0.07356736 0.21582806 0.06572546]
 [0.0656609  0.4675519  0.46131896]]
```

```python
def fill_null_values(X, y, name, means = None):
    """...
    df = pd.concat([X, y], axis = 1)

    if means is None:
        means = X.mean()

    # Replace null values with the average (mean) of the column
    X = X.fillna(means)

    # Statistics of the data set.
    stats = df.describe()

    # Confirm that all columns have the same amount of data and that there are no null values.
    first_column_count = stats.loc['count'].iloc[0]
    null_values = df.isnull().any().any()
    print(f'\nThe total amount of data for the {name} is equal to {first_column_count} in all columns.')
    print(f'It is {null_values} that there are null values, and the means of the inputs in the {name} is')
    X_mean_table = X.mean().to_numpy().reshape(7, 3)
    print(X_mean_table)

    return X, means
```

ACCESS AND PREPROCESS THE DATA

```
X_train, y_train = remove_outliers(X_train, y_train,'Training set')
X_test, y_test = remove_outliers(X_test, y_test, 'Testing set')
# X_validation, y_validation = remove_outliers(X_validation, y_validation, 'Validation set') DON'T REMOVE
X, y = remove_outliers(X, y, 'Total data set')
```

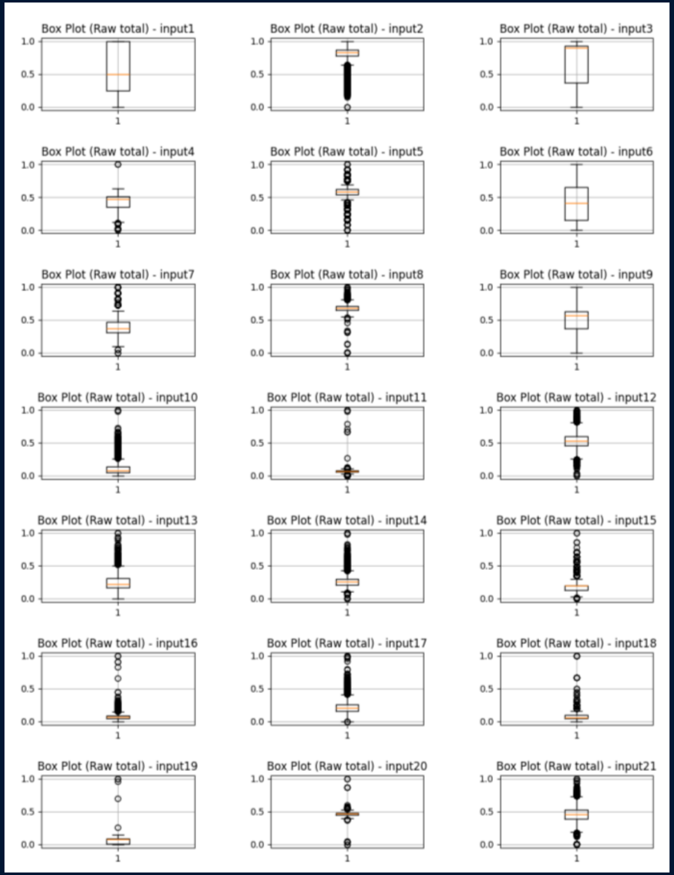
✓ 1.5s

```
806 rows have been removed from Training set, which is the 9.788681078455186% percent of the original data.
116 rows have been removed from Testing set, which is the 7.047387606318348% percent of the original data.
1093 rows have been removed from Total data set, which is the 9.955369341470078% percent of the original data.
```

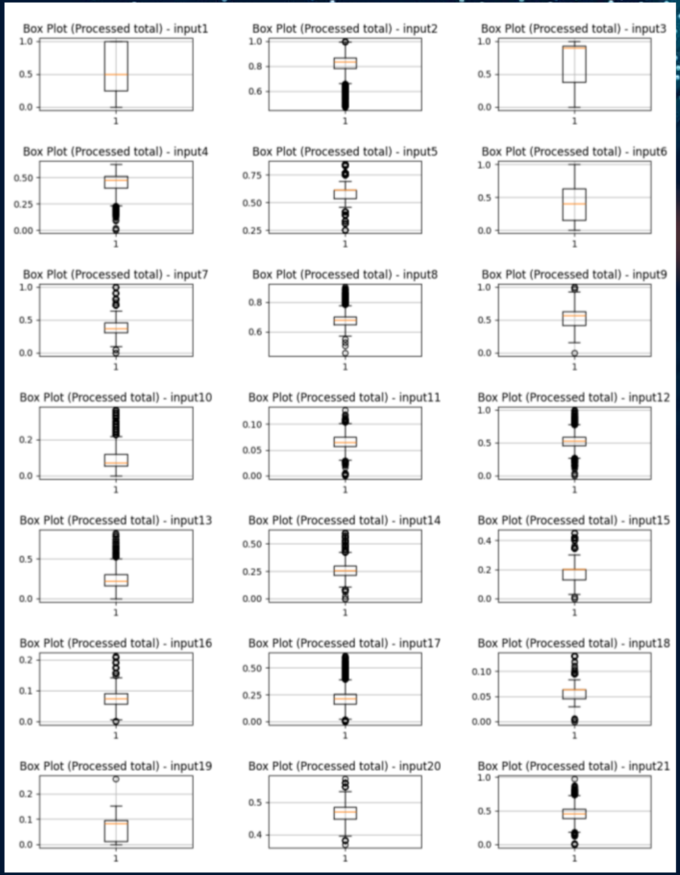
Remove outliers

```python
def remove_outliers(X, y, name, threshold = 3.75):
    """...
    df = pd.concat([y, X], axis = 1)

    input_columns = X.columns
    rows_before = len(df)

    for i in range(5):
        for column in input_columns:

            Q1 = df[column].quantile(0.25)
            Q3 = df[column].quantile(0.75)
            IQR = Q3 - Q1

            lower_bound = Q1 - threshold * IQR
            upper_bound = Q3 + threshold * IQR

            outliers = (df[column] < lower_bound) | (df[column] > upper_bound)
            df = df[~outliers]

    rows_after = len(df)
    rows_removed = rows_before - rows_after
    percentage_removed = (rows_removed/rows_before)*100
    print(f'{rows_removed} rows have been removed from {name}, which is the {percentage_removed}% percent of the original data.')

    return df.iloc[:, 1:], df.iloc[:, 0]
```

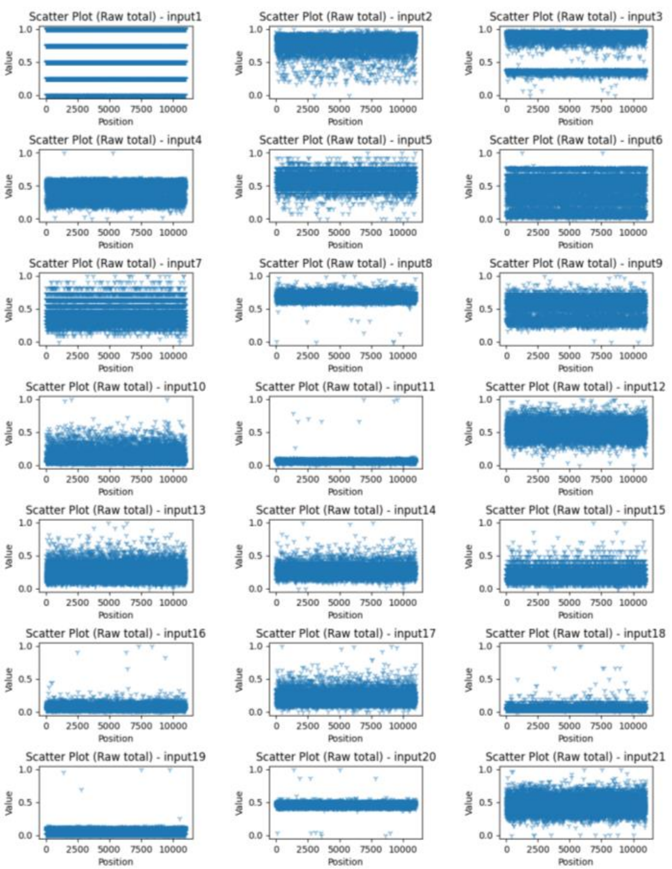
EXPLORATORY DATA ANALYSYS THROUGH VISUALIZATION



Box plots

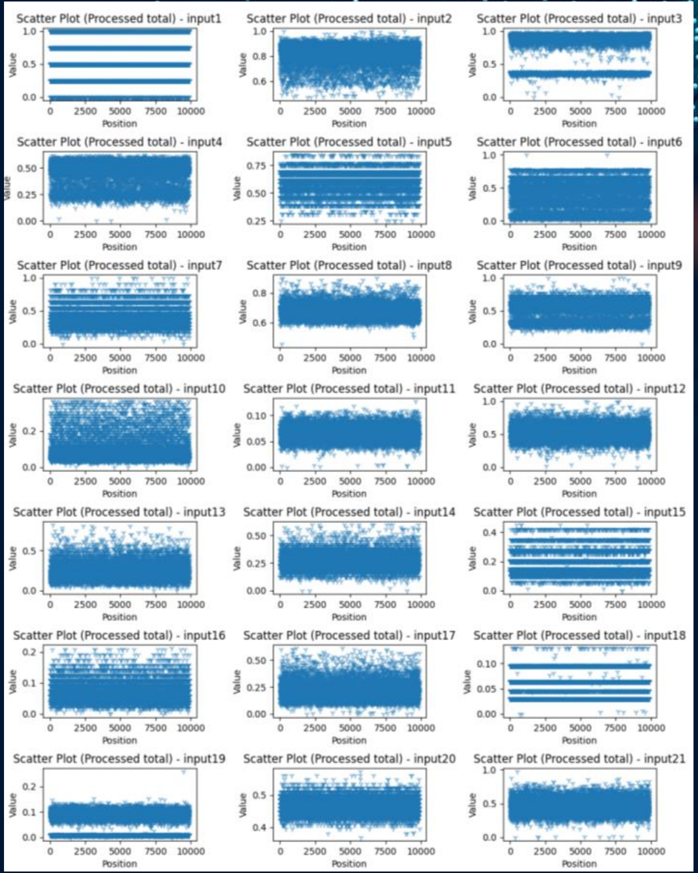


EXPLORATORY DATA ANALYSYS THROUGH VISUALIZATION

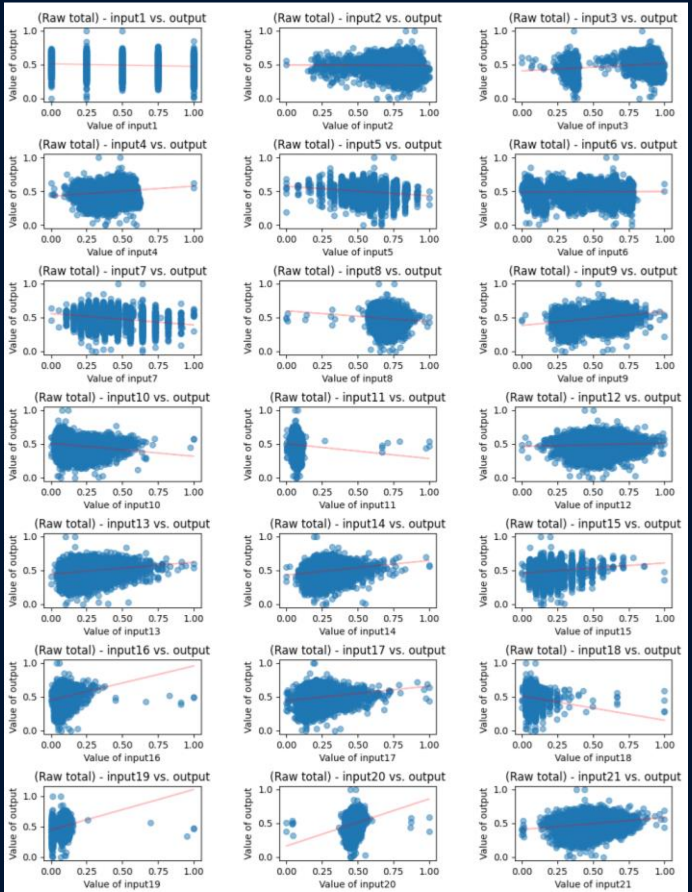


Scatter plots

(just inputs)

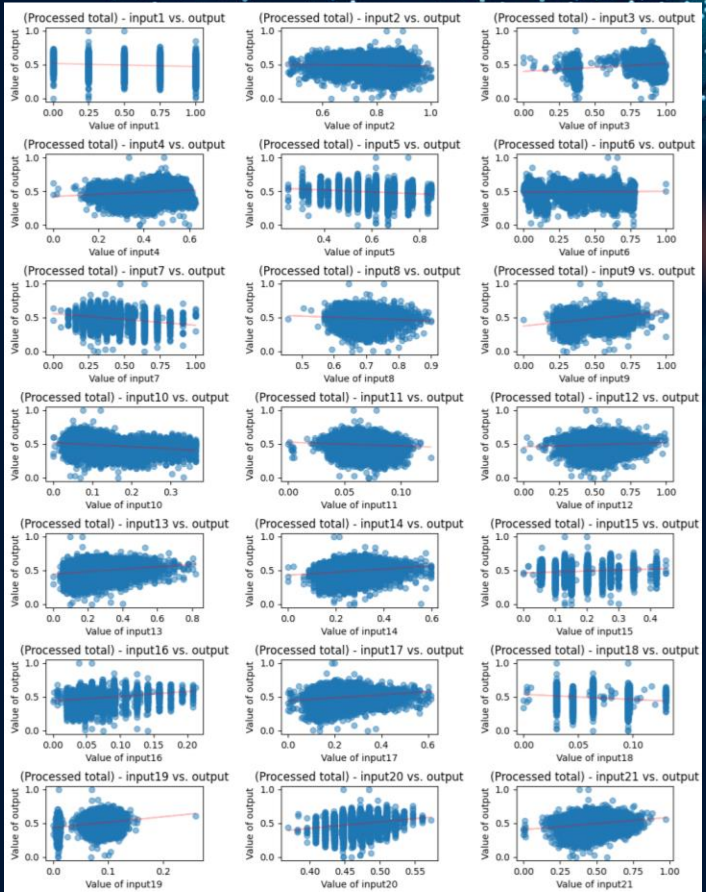


EXPLORATORY DATA ANALYSYS THROUGH VISUALIZATION

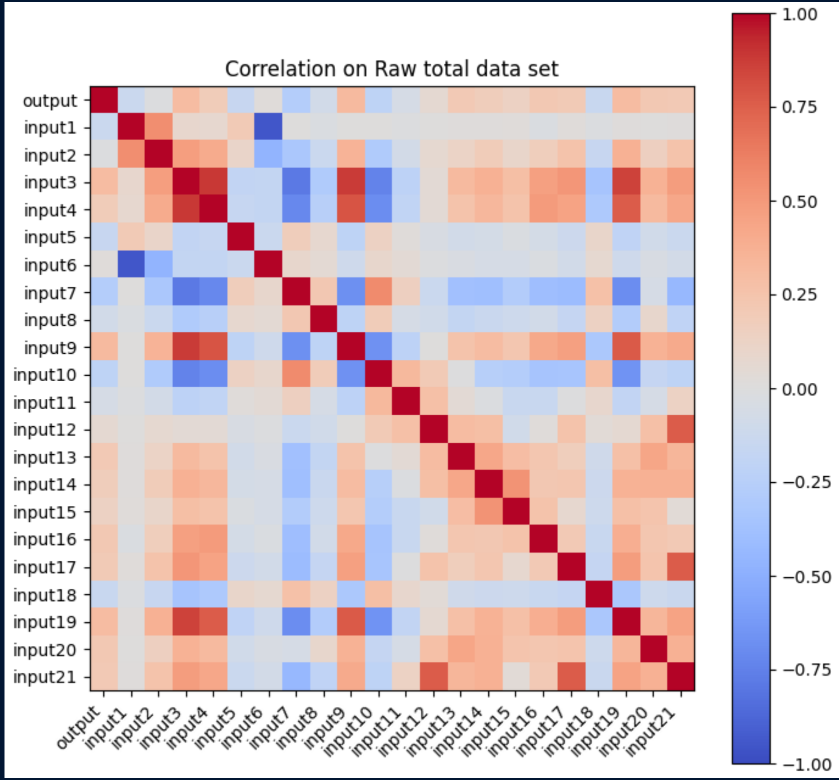


Scatter plots

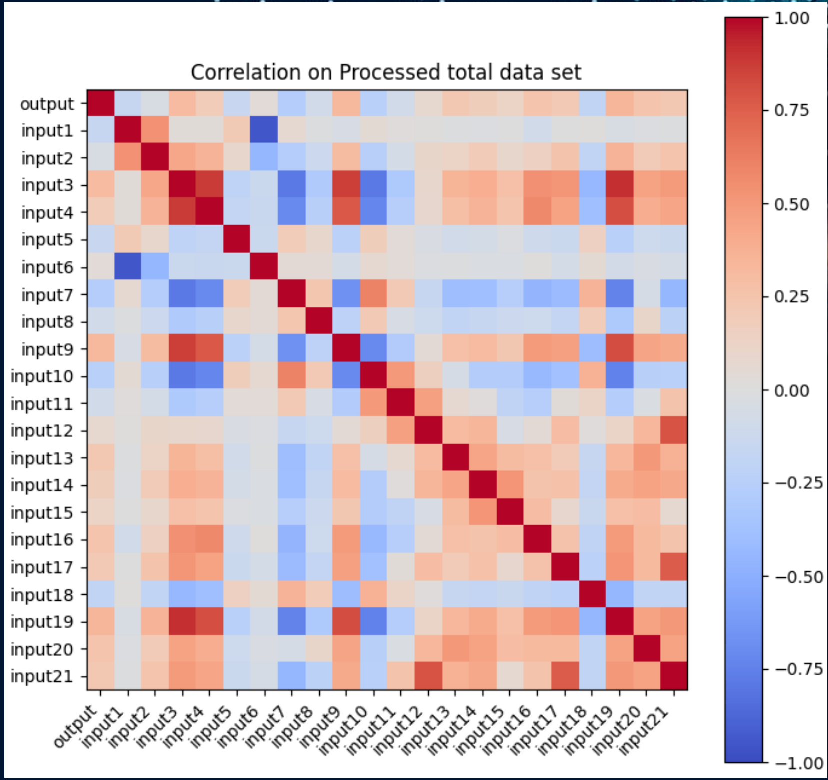
(inputs x output)



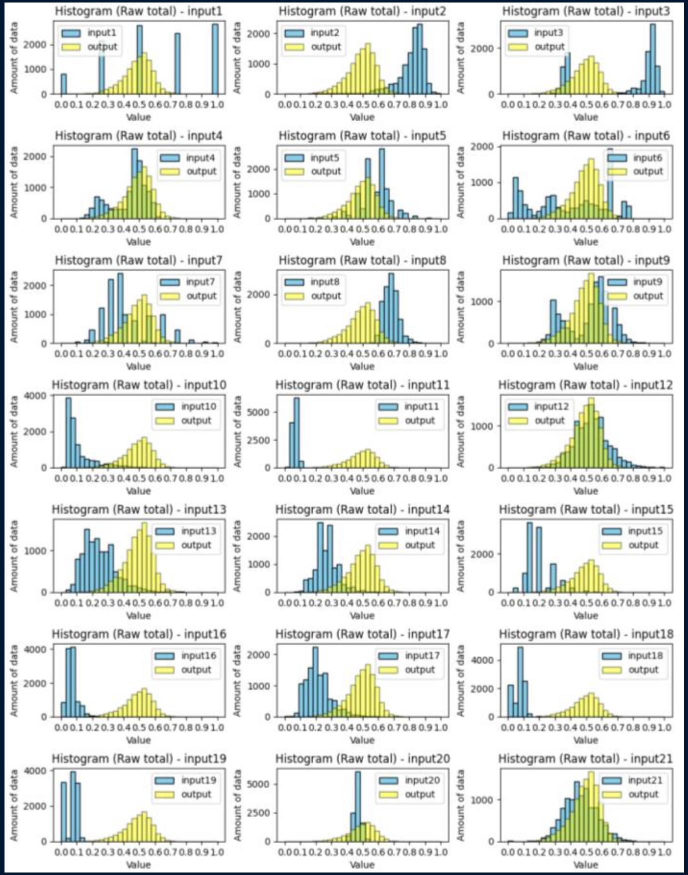
EXPLORATORY DATA ANALYSYS THROUGH VISUALIZATION



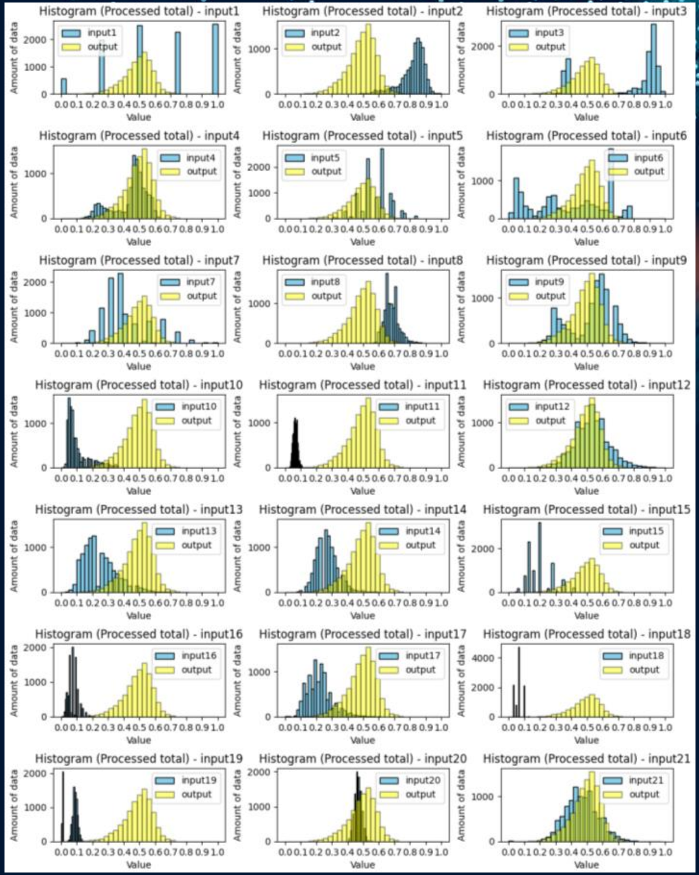
Heat maps



EXPLORATORY DATA ANALYSYS THROUGH VISUALIZATION



Histograms



CROSS VALIDATION AND HYPER-PARAMETERS

GridSearch

```python
def train_model(model_name, model, param_grid, X_train, y_train, X_test, y_test, X_val, y_val, results_df, nn = False):
    """...
    start_time = time.time()

    grid_search = GridSearchCV(model, param_grid, n_jobs = -1, scoring = 'neg_mean_squared_error', cv = 10)
    grid_search.fit(X_train, y_train)
    print_results(grid_search, nn)

    end_time = time.time()
    training_duration = (end_time - start_time)/60

    best_model = grid_search.best_estimator_
    best_mean = -grid_search.best_score_
    best_std = np.sqrt(grid_search.cv_results_['std_test_score'][grid_search.best_index_])

    y_pred_test = best_model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_pred_test)

    y_pred_validation = best_model.predict(X_val)
    mse_validation = mean_squared_error(y_val, y_pred_validation)

    print(f'Mean Squared Error on Test dataset (df_testing_data): {mse_test}')
    print(f'Mean Squared Error on Validation dataset (df_validation_data): {mse_validation}')

    results_df.loc[len(results_df)] = {
        'Model': model_name,
        'Mean': best_mean,
        'Standard deviation': best_std,
        'MSE on Testing set': mse_test,
        'MSE on Validation set': mse_validation,
        'Training duration': training_duration,
        'Parameters': grid_search.best_params_
    }
    return best_model, results_df
```

```python
def print_results(model, neural_network):
    """...
    print('All results:')
    means = model.cv_results_['mean_test_score']
    stds = model.cv_results_['std_test_score']
    params = model.cv_results_['params']
    for mean, std, params in zip(means, stds,params):
        print('Mean = %0.3f and Standard deviation = +/-%0.03f for %r.' % (mean, std*2, params))

    print('Best parameters found:\n', model.best_params_)

    best_model = model.best_estimator_

    if (neural_network):
        n_iter = best_model.n_iter_
        print('Number of iterations for convergence:', n_iter)
```

MODEL ARCHITECTURE - MACHINE LEARNING ALGORITHMS

Results performance models

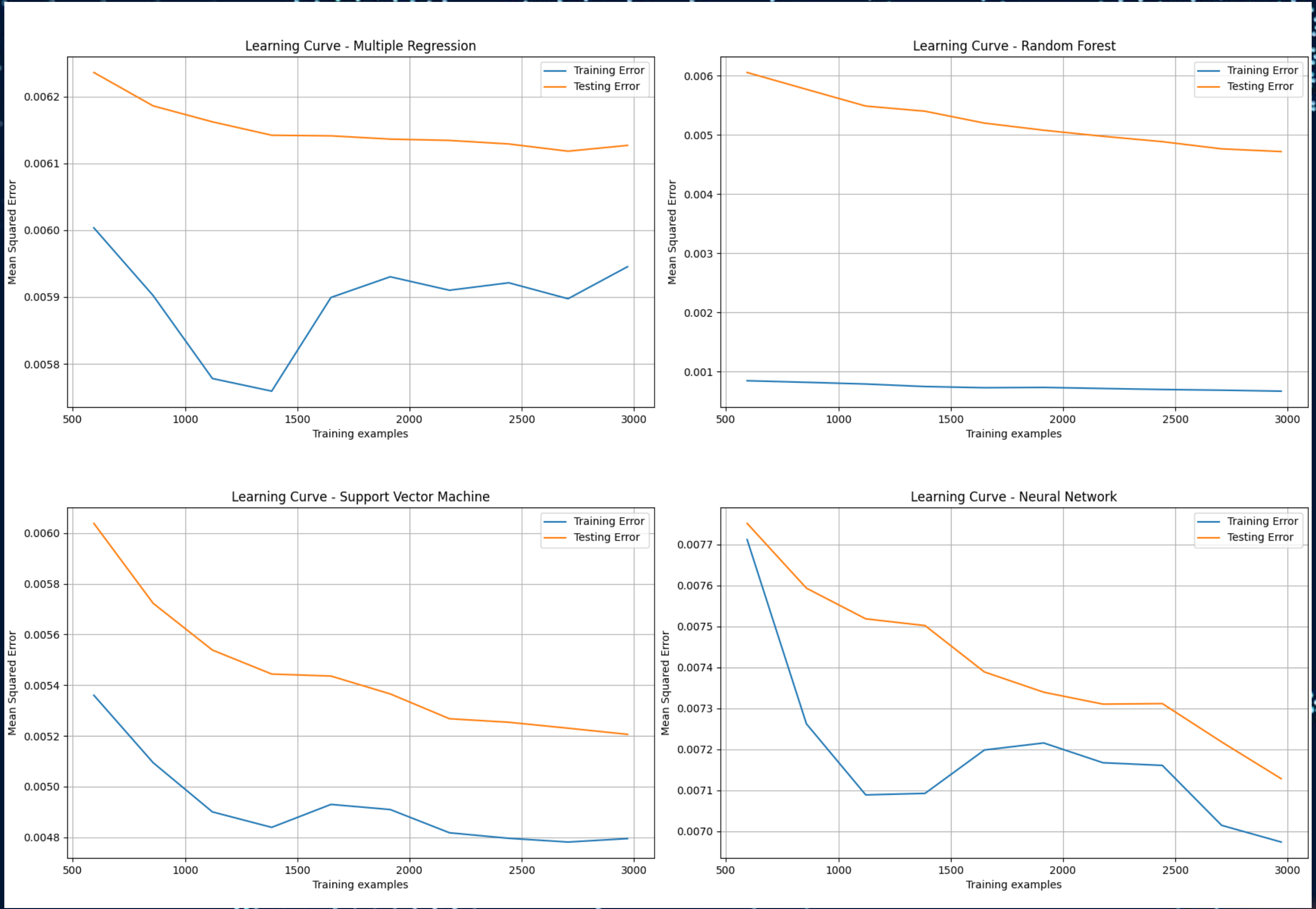
Model	Mean	Standard deviation	MSE on Testing set	MSE on Validation set	Training duration	Parameters
Multiple Regression	0.006093228	0.01857774	0.006287932	0.006485017	0.200452642	{}
Random Forest	0.004141866	0.020015752	0.004286809	0.004519707	42.47799547	{'max_depth': 50, 'n_estimators': 500}
Support Vector Machine	0.00498062	0.021461261	0.005248416	0.005097594	22.84293645	{'C': 5.0, 'gamma': 'scale', 'kernel': 'rbf'}
Neural Network	0.005978933	0.020237692	0.006238387	0.006425752	94.82012931	{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': 250, 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_iter': 500, 'solver': 'adam', 'tol': 0.0001}

01 MULTIPLE REGRESSION

02 RANDOM FOREST

03 SUPPORT VECTOR MACHINE

04 NEURAL NETWORK



CONCLUSIONS!

- All machine learning models demonstrate sufficient efficiency and functionality. However, in general terms, the current preferred choices are Random Forest as the top option, followed by Support Vector Machine, because they exhibited the smallest error.

- If a substantial amount of additional data were to become available, the Neural Network might become the optimal choice. As evident from its learning curve, there is still potential for further improvement and error reduction. However, due to the limited amount of data available, the Neural Network did not achieve its full training potential.

THANKS!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.